

(11)Publication number : 2004-013190
(43)Date of publication of application : 15.01.2004

2006/02/14

(19) 日本国特許庁(JP)

(12) 公開特許公報(A)

(11) 特許出願公開番号

特開2004-13190

(P2004-13190A)

(43) 公開日 平成16年1月15日(2004.1.15)

(51) Int. Cl.⁷
G06F 9/45F 1
G06F 9/44 322Eテーマコード (参考)
5B081

審査請求 未請求 請求項の数 17 O L (全 19 頁)

(21) 出願番号 特願2002-161486 (P2002-161486)
(22) 出願日 平成14年6月3日 (2002.6.3)(71) 出願人 000005821
松下電器産業株式会社
大阪府門真市大字門真1006番地
(74) 代理人 100097179
弁理士 平野 一幸
(72) 発明者 近藤 孝宏
大阪府門真市大字門真1006番地 松下
電器産業株式会社内
(72) 発明者 中村 剛
大阪府門真市大字門真1006番地 松下
電器産業株式会社内
(72) 発明者 樽木 麻衣子
大阪府門真市大字門真1006番地 松下
電器産業株式会社内
Fターム (参考) 5B081 CC41

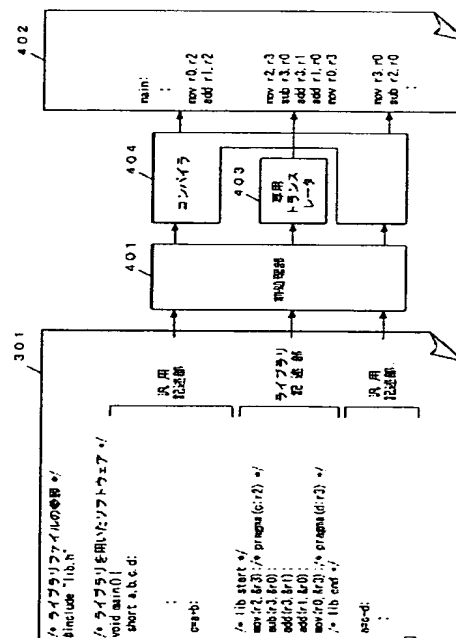
(54) 【発明の名称】 ソフトウェア開発環境、シミュレータ及び記録媒体

(57) 【要約】

【課題】評価ボード等が必要なく、かつ、従来技術よりも高速な検証が行える技術を提供する。

【解決手段】ソースコードは、全て高級言語で記述され、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数等からなるライブラリ記述部と、ライブラリ記述部以外の汎用記述部とを含む。ターゲットプロセッサ用のソフトウェアを生成するには、ライブラリ記述部を、専用トランスレータ403でアセンブラコードに、一対一に変換し、汎用記述部を、コンパイラ404によりコンパイルする。シミュレータのコンパイラ906は、ライブラリ記述部を、ライブラリを参照してコンパイルする。

【選択図】 図4



【特許請求の範囲】

【請求項1】

高級言語で記述されたソースコードを読み込んで、ターゲットプロセッサ用のアセンブラコードを生成すると共に、このターゲットプロセッサとは異なるホストプロセッサ上で動作するコンパイラを備え、

前記ソースコードは、

ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、

高級言語で記述され、かつ前記ライブラリ記述部に該当しない汎用記述部とを含む、ソフトウェア開発環境。

10

【請求項2】

前記ライブラリ記述部は、専用トランスレータによりターゲットプロセッサ用のアセンブラコードに、一対一に変換され、かつ、前記汎用記述部は、前記コンパイラによりコンパイルされて、ターゲットプロセッサ用のアセンブラコードが生成される、請求項1記載のソフトウェア開発環境。

【請求項3】

ソースコードは、変数の割付に関する情報を含み、かつ前記コンパイラは、この情報を反映した、ターゲットプロセッサ用のアセンブラコードを生成する、請求項2記載のソフトウェア開発環境。

【請求項4】

ソースコードにおいて、前記ライブラリ記述部と前記汎用記述部とは、前記コンパイラによりコメントと解釈される情報により区別されている、請求項1から3記載のソフトウェア開発環境。

20

【請求項5】

ソースコードにおいて、前記ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、前記汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、前記コンパイラによりコメントと解釈される書式で記述されている、請求項1から4記載のソフトウェア開発環境。

【請求項6】

変数の割付に関する情報は、前記コンパイラによりコメントと解釈される書式で記述されている、請求項3記載のソフトウェア開発環境。

30

【請求項7】

ソースコードには、前記ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている請求項1から6記載のソフトウェア開発環境。

【請求項8】

請求項1から7記載のソフトウェア開発環境を実現するプログラムが記録された記録媒体

【請求項9】

高級言語で記述されたソースコードを読み込んで、このソースコードによる、ターゲットプロセッサの動作をシミュレートし、かつ、ホストプロセッサ上で動作するオブジェクトコードを生成するコンパイラを備え、

40

それ自体ホストプロセッサ上で動作する、シミュレータであって、

前記ソースコードは、

ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、

高級言語で記述され、かつ前記ライブラリ記述部に該当しない汎用記述部とを含み、

前記ライブラリ記述部で使用される関数又は手続きの定義を含むライブラリが、前記コンパイラにより参照可能に用意されており、

前記コンパイラは、前記汎用記述部をコンパイルすると共に、前記ライブラリ記述部を、

50

このライブラリを参照してコンパイルして、ホストプロセッサ用のアセンブラコードを生成し、生成されたアセンブラコードを、ホストプロセッサ上で実行可能なオブジェクトコードに変換する、シミュレータ。

【請求項 10】

前記ライブラリ記述部は、変数の割付に関する情報を含み、この情報に基づいて、前記ライブラリ記述部と前記汎用記述部の接続部を生成し、前記オブジェクトコードに付加する、請求項 9 記載のシミュレータ。

【請求項 11】

前記ライブラリには、ターゲットプロセッサの割り込み機能が備えられ、ホストプロセッサ上でのシミュレーション時に、この割り込み機能のオン/オフを制御可能とした、請求項 9 から 10 記載のシミュレータ。

【請求項 12】

前記ライブラリには、ターゲットプロセッサの実行サイクル数カウント機能が備えられ、ホストプロセッサ上でのシミュレーション時にターゲットプロセッサのサイクル数計測を可能とした、請求項 9 から 11 記載のシミュレータ。

【請求項 13】

ソースコードにおいて、前記ライブラリ記述部と前記汎用記述部とは、前記コンパイラによりコメントと解釈される情報により区別されている、請求項 9 から 12 記載のシミュレータ。

【請求項 14】

ソースコードにおいて、前記ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、前記汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、前記コンパイラによりコメントと解釈される書式で記述されている、請求項 9 から 13 記載のシミュレータ。

【請求項 15】

変数の割付に関する情報は、前記コンパイラによりコメントと解釈される書式で記述されている、請求項 10 記載のシミュレータ。

【請求項 16】

ソースコードには、前記ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている請求項 9 から 15 記載のシミュレータ。

【請求項 17】

請求項 9 から 16 記載のシミュレータを実現するプログラムが記録された記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、ターゲットプロセッサ向け組み込みソフトウェアの開発に適した、ソフトウェア開発環境及びその関連技術に関するものである。

【0002】

【従来の技術】

ターゲットプロセッサ向け組み込みソフトウェアを開発する際、一般に、ターゲットプロセッサとは互換性のない、ホストプロセッサ上で動作するソフトウェア開発環境が用いられる。

【0003】

ここで、本明細書において、「ホストプロセッサ」とは、このソフトウェア開発環境を動作させ、開発と、開発の結果物であるソフトウェアの検証とに、使用するプロセッサをいう。

【0004】

また、「ターゲットプロセッサ」とは、ホストプロセッサとは異なるプロセッサであり、

10

20

30

40

50

この開発の結果物であるソフトウェアを実行するプロセッサをいう。

【0005】

そして、ホストプロセッサとターゲットプロセッサとは、ソフトウェア互換性がない。また、このような、結果物としてのソフトウェアは、専らターゲットプロセッサ上でのみ正常に動作し、ホストプロセッサ上では正常に動作しない。

【0006】

また、本明細書にいう「シミュレータ」が生成するソフトウェアは、上記結果物であるソフトウェアを模したソフトウェアであるが、ターゲットプロセッサ上で動作するものではなく、ホストプロセッサ上で動作するものである。

【0007】

このように、ホストプロセッサのソフトウェア開発環境を用いて、ホストプロセッサとは異なるターゲットプロセッサ向けのソフトウェアを開発する場合、次に述べるような問題点がある。

【0008】

ここで近年、このようなソフトウェアを開発する場合であっても、ソフトウェアが大規模になるに伴い、アセンブラ言語主体による開発から、C/C++等の高級言語による開発へ、シフトしている。

【0009】

これは、高級言語を用いたソフトウェア開発を行えば、データの保持、転送、演算等の処理を、ターゲットプロセッサのアセンブラレベルの命令やレジスタ、メモリ等のリソースに依存しない形式で記述でき、可読性、汎用性、開発効率に優れているためである。

【0010】

ここで特に、組み込み向けソフトウェア等では、システムのパフォーマンスを最大限引き出すため、プロセッサの能力を極限まで高める、最適化が求められる。

【0011】

しかしながら、高級言語によるソフトウェア開発では、コンパイラ性能の問題から高級言語からアセンブラコードに変換する際に、冗長なコードが生成され、ソフトウェアのコードサイズや実行速度に影響を及ぼす場合がある。

【0012】

そこで現在でも、信号処理などの処理負荷が大きい部分に関しては、高級言語による開発に加え、ターゲットプロセッサのアセンブラ言語による開発が行われている。

【0013】

そのためソフトウェア開発者は、図11(a)のソースコード例1101が示すように、多くのコンパイラがサポートしている、高級言語とアセンブラ言語の混在記述仕様“asm文”等を用いたり、図11(b)に示すように、高級言語によるソースコード例1102の他に、同図のアセンブラ例1103に示すように、プログラムの一部(例えば一つの関数)を、アセンブラのみで記述する開発方法をとっている。

【0014】

そして、このように記述されたソースコードを、コンパイラを用いてアセンブラコードに翻訳する。このとき、“asm文”等やアセンブラで記述された部分についても、コンパイラに解釈され、コンパイラは、アセンブラで記述された部分については、アセンブラコードをそのまま出力する。

【0015】

そして、アセンブラコードをターゲットプロセッサ上で実行可能なオブジェクトコードに変換する。上述したように、このオブジェクトコードは、ターゲットプロセッサ上でのみ動作し、ホストプロセッサ上では動作しない。

【0016】

また、ソフトウェアの開発を進めるには、結果物であるソフトウェアが、要求されている仕様を満たすか否か、検証を行う必要がある。ここで、ホストプロセッサ上で、結果物であるソフトウェア(ターゲットプロセッサ用のソフトウェアそのもの)を動作させること

10

20

30

40

50

ができれば、話は簡単であるが、上述のように、これはできない。したがって、その他の検証法によりざるを得ない。

【0017】

従来技術において、この検証を行うには、(検証法1) ターゲットプロセッサ上で実際に実行させる、または、(検証法2) ホストプロセッサ上で動作するシミュレータにより、ターゲットプロセッサの動作をシミュレートする、という手法が、主にとられる。

【0018】

【発明が解決しようとする課題】

ここで、(検証法1) により、結果物であるソフトウェアを、ターゲットプロセッサ上で実行させようとするには、ターゲットプロセッサ等を物理的に実装した評価ボード等を、ソフトウェア開発者の数だけ用意する必要があり、コスト的に課題がある。また、このような評価ボード等を入手できないときには、この検証法では、全く検証を行えない。

10

【0019】

また、(検証法2) により、ホストプロセッサ上で動作するシミュレータを用いると、シミュレーション速度が遅い、という課題がある。

【0020】

これは、従来のこの種のシミュレータは、インタープリタ型のものしかなく、このシミュレータは、実行するソフトウェアをメモリ内に格納し、実行時に、1命令ずつメモリから読み出し、その命令を解釈し、実行するという、プロセスを経るからである。

【0021】

そこで本発明は、評価ボード等が必要なく、かつ、従来技術よりも高速な検証が行える技術を提供することを目的とする。

20

【0022】

より具体的には、本発明は、この目的に合致し、ターゲットプロセッサ用のアセンブラコードを生成する、ソフトウェア開発環境を提供する。また本発明は、この目的に合致し、ホストプロセッサ上で動作するオブジェクトコードを生成する、コンパイラ型のシミュレータを提供する。

【0023】

【課題を解決するための手段】

第1の発明に係るソフトウェア開発環境は、高級言語で記述されたソースコードを読み込んで、ターゲットプロセッサ用のアセンブラコードを生成すると共に、このターゲットプロセッサとは異なるホストプロセッサ上で動作するコンパイラを備え、ソースコードは、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、高級言語で記述され、かつライブラリ記述部に該当しない汎用記述部とを含む。

30

【0024】

この構成により、ソフトウェア開発環境と、シミュレータとに、共通のソースコードを適用できる。また、ソースコードが全て高級言語で記述されるため、移植性や可読性に優れた、開発を行える。

【0025】

第2の発明に係るソフトウェア開発環境では、ライブラリ記述部は、専用トランスレータによりターゲットプロセッサ用のアセンブラコードに、一対一に変換され、かつ、汎用記述部は、コンパイラによりコンパイルされて、ターゲットプロセッサ用のアセンブラコードが生成される。

40

【0026】

この構成により、ライブラリ記述部における、アセンブラレベルでのコード最適化を、生成するアセンブラコードに、そのまま反映できる。

【0027】

第3の本発明に係るソフトウェア開発環境では、ソースコードは、変数の割付に関する情報を含み、かつコンパイラは、この情報を反映した、ターゲットプロセッサ用のアセン

50

ラコードを生成する。

【0028】

この構成により、変数の割付に関する指定を、アセンブラコードに反映できる。

【0029】

第4の発明に係るソフトウェア開発環境では、ソースコードにおいて、ライブラリ記述部と汎用記述部とは、コンパイラによりコメントと解釈される情報により区別されている。

【0030】

この構成において、ライブラリ記述部は、コンパイラによりコメントと解釈される情報により、汎用記述部と区別されているため、この区別のための情報は、コンパイラにより無視され、コンパイラ型のシミュレータによるシミュレーション時と、ソースコードを共通にしても、不都合を生じない。

10

【0031】

第5の発明に係るソフトウェア開発環境では、ソースコードにおいて、ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、コンパイラによりコメントと解釈される書式で記述されている。

【0032】

この構成において、ライブラリ記述部は、開始識別子と終了識別子とに挟まれているため、ライブラリ記述部と、汎用記述部とを、明確に区別できる。

【0033】

第6の発明に係るソフトウェア開発環境では、変数の割付に関する情報は、コンパイラによりコメントと解釈される書式で記述されている。

20

【0034】

この構成において、変数の割付に関する情報が、コンパイラによりコメントと解釈される書式で記述されているため、この記述は、コンパイラにより無視され、コンパイラ型のシミュレータによるシミュレーション時と、ソースコードを共通にしても、不都合を生じない。

【0035】

第7の発明に係るソフトウェア開発環境では、ソースコードには、ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている。

30

【0036】

この構成により、シミュレーション時のソースコードと、完全なコード共通性を確保できる。なお、ソースコードに、このコンパイラ疑似命令を含めても、コンパイラの処理上、不都合を生じない。

【0037】

第8の発明に係るシミュレータは、高級言語で記述されたソースコードを読み込んで、このソースコードによる、ターゲットプロセッサの動作をシミュレートし、かつ、ホストプロセッサ上で動作するオブジェクトコードを生成するコンパイラを備え、それ自体ホストプロセッサ上で動作する、シミュレータであり、ソースコードは、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、高級言語で記述され、かつライブラリ記述部に該当しない汎用記述部とを含み、ライブラリ記述部で使用される関数又は手続きの定義を含むライブラリが、コンパイラにより参照可能に用意されており、コンパイラは、汎用記述部をコンパイルすると共に、ライブラリ記述部を、このライブラリを参照してコンパイルして、ホストプロセッサ用のアセンブラコードを生成し、生成されたアセンブラコードを、ホストプロセッサ上で実行可能なオブジェクトコードに変換する。

40

【0038】

この構成により、コンパイラ型のシミュレータを実現できる。しかも、このソースコードは、ソフトウェア開発環境に用いるものと、共通のものを利用できる。コンパイラ型のシ

50

ミュレータであるから、従来のインタープリタ型のシミュレータに比べ、遙かに高速なオブジェクトコードが得られ、効果的な検証を行える。

【0039】

第9の発明に係るシミュレータでは、ライブラリ記述部は、変数の割付に関する情報を含み、この情報に基づいて、ライブラリ記述部と汎用記述部の接続部を生成し、オブジェクトコードに付加する。

【0040】

この構成により、変数の整合性を確保したオブジェクトコードを、生成できる。

【0041】

第10の発明に係るシミュレータでは、ライブラリには、ターゲットプロセッサの割り込み機能が備えられ、ホストプロセッサ上でのシミュレーション時に、この割り込み機能のオン/オフを制御可能とした。

10

【0042】

この構成により、割り込み機能が必要な場合は当該機能を使用したシミュレーションを行い、割り込み機能が不要なシミュレーションの場合は高速なシミュレーションを行える。

【0043】

第11の発明に係るシミュレータでは、ライブラリには、ターゲットプロセッサの実行サイクル数カウント機能が備えられ、ホストプロセッサ上でのシミュレーション時にターゲットプロセッサのサイクル数計測を可能とした。

【0044】

この構成により、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照できる。

20

【0045】

【発明の実施の形態】

以下、図面を参照しながら、本発明の実施の形態を説明する。なお以下、高級言語として、C言語を取り上げるが、Pascal等他の高級言語を適用することもできる。

(実施の形態1)

以下、本発明の実施の形態1について、図1～図4を参照しながら説明する。本形態は、ソフトウェア開発環境に関する。

【0046】

図1(a)は、本発明の実施の形態1で取り上げるターゲットプロセッサのブロック図である。

30

【0047】

図1(a)に示すように、本ターゲットプロセッサは、リソースとして4本の16ビットレジスタr0、r1、r2、r3を持ち、演算器として、2つの符号付16ビットデータを入力し、その加算/減算結果である符号付16ビットデータを出力する、加減算器101を備える。

【0048】

ここで、加減算器101は、演算結果が、符号付16ビットの最大値(0x7FFF)よりも大きい値になった場合は、演算結果を符号付16ビットの最大値(0x7FFF)とし、演算結果が、符号付16ビットの最小値(0x8000)よりも小さい値になった場合は、符号付16ビットの最小値(0x8000)にクリップする、MAXMIN機能を、装備しているものとする。

40

【0049】

なお、本形態では、ターゲットプロセッサとして、DSPを想定しているため、ターゲットプロセッサは、DSPではあたりのMAXMIN機能を、持つことを前提としているが、本発明が予定するターゲットプロセッサとしては、MAXMIN機能を有するものに限定されない。

【0050】

さて、加減算器101と、4本のレジスタr0～r3とは、バス102、103、104

50

により接続され、4本のレジスタr0～r3のうちの2本は、加減算器101への入力レジスタとして使用され、1本は、加減算器101からの結果を格納する出力レジスタとして使用される。なお、入力レジスタ及び出力レジスタとして、どのレジスタを使用するかは、後述するアセンブラ命令により指定される。

【0051】

図1(b)は、同ターゲットプロセッサ上で実行可能なアセンブラレベルの命令及びその機能を示す説明図である。

【0052】

図1(b)に示すように、本ターゲットプロセッサは、3つのアセンブラ命令、“mov命令”、“add命令”、“sub命令”に対応している。勿論、対応する命令を適宜変更しても、本発明に包含される。

10

【0053】

ここで、これら各命令の記述形式及び機能について説明する。まず、転送を行う、“mov命令”は“mov src, dst”と記述する。ここで、“src”及び“dst”は、4本のレジスタr0～r3から選択する。例えば“mov r0, r1”といった具合である。この命令によって、レジスタr0に格納されているデータが、レジスタr1に転送される。

【0054】

次に、加算を行う、“add命令”は“add src, dst”と記述する。ここで、“src”及び“dst”は、“mov命令”と同様である。例えば、“add r0, r1”という命令では、レジスタr0、r1に格納されたデータが、加算され、結果がレジスタr1に格納される。

20

【0055】

また、減算を行う、“sub命令”は“sub src, dst”と記述する。ここで“src”及び“dst”は、“mov命令”、“add命令”と同様である。例えば、“sub r0, r1”という命令では、レジスタr0に格納されたデータからレジスタr1に格納されたデータが減算され、結果がレジスタr1に格納される。

【0056】

図2は、図1のターゲットプロセッサに対応するライブラリをC言語で記述した例示図である。本例では、このライブラリは、“lib.h”というファイル名を持つ1つのファイルからなるが、複数のファイルに分けてライブラリを記述することもできる。

30

【0057】

図2に示すように、本形態で使用するライブラリは、ターゲットプロセッサのリソースである4本のレジスタ(r0、r1、r2、r3)の機能を実現する変数(r0、r1、r2、r3)の定義と、ターゲットプロセッサが対応している各アセンブラ命令(mov命令、add命令、sub命令)の機能を実現する、C言語の関数(mov関数、add関数、sub関数)の定義とを、含む。

【0058】

まず、ターゲットプロセッサのレジスタ(r0～r3)は、16ビットであるため、ライブラリにおいて、これらのレジスタに対応する、変数(r0～r3)は、16ビットデータ型“short”で宣言する。また、ライブラリにおいて、それ以外の演算精度等も、ターゲットプロセッサと合致するように記述する。

40

【0059】

またここで、各アセンブラ命令(mov命令、add命令、sub命令)と、ライブラリにおいて、これらの命令に対応するC言語の関数(mov関数、add関数、sub関数)とは、一対一に対応している。なお、引数の内容及び順序も、一致させておくことが望ましい。

【0060】

そして、これらC言語の関数は、引数に演算に使用するレジスタ(変数r0～r3)を指定して呼び出され、指定されたレジスタに格納されたデータに対して各種演算を行い、結

50

果を変数にセットして戻り構成となっている。

【0061】

ターゲットプロセッサにおいて、レジスタ間のデータ転送命令であるmov命令機能を実現するmov関数は、第1の引数で渡された転送元のレジスタ値(変数src値)を第2の引数で渡されたレジスタ(変数*dst)へ転送する。

【0062】

また、加算命令であるadd命令機能を実現するadd関数は、第1及び第2の引数で渡された入力レジスタ値(変数src、*dst値)に対し加算処理並びにMAXMIN処理を実行し、結果を第2の引数で渡されたレジスタ(変数*dst)に格納する。

【0063】

減算命令であるsub命令機能を実現するsub関数も、同様である。

【0064】

なお、C言語の仕様上、戻り値のない(void型)の処理も、“手続き”とは呼ばず、“関数”と呼ぶ習慣があるため、図2のライブラリは、変数と関数の定義からなる。

【0065】

しかしながら、他の高級言語、例えばPascalでは、戻り値のない手続き(Procedure)と、戻り値を持つ関数(function)とを、厳格に区別している。また、本明細書にいう高級言語には、C言語以外的高级言語、例えばPascal等も、含めている。したがって、本明細書にいうライブラリでは、アセンブラコードに対応する処理は、一般に、“関数又は手続きにより高級言語で定義されている”ということになる。

【0066】

図3は、本発明の実施の形態1におけるソースコードの例示図である。以下、ソースコードにおいて、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された、関数又は手続きで表現される部分を「ライブラリ記述部」といい、ライブラリ記述部以外の部分を「汎用記述部」という。

【0067】

図3に示すように、このソースコード301では、まず先頭で、ライブラリファイル(“lib.h”)が、インクルードされる。

【0068】

また、ライブラリ記述部では、このライブラリにおいて定義された、変数や関数を用いて、プログラムが記述される。

【0069】

図3において、“/* lib_start */”は、ライブラリ記述部の開始識別子であり、“/* lib_end */”は、ライブラリ記述部の終了識別子である。

【0070】

このように、本形態では、ライブラリ記述部は、開始識別子と終了識別子とで挟まれることにより、汎用記述部と、明確に区別される。

【0071】

しかも、これら開始識別子及び終了識別子は、いずれもC言語のコンパイラにより、コメントと解釈される書式となっている。さらに、ライブラリ記述部には、“asm文”等の標記は、行われていない。

【0072】

よって、このソースコード301を、C言語のコンパイラに通すと、汎用記述部だけでなく、ライブラリ記述部についても、コンパイラは、C言語そのもの(asm文等は除く)で記述されていると解釈する。

【0073】

より具体的には、コンパイラが、“mov(r2, &r3);”という行を処理するとき、これをC言語の関数として処理し、文頭でインクルードされている、ライブラリファイル“lib.h”にある、mov関数の定義を、この行に、あてはめる。

【0074】

10

20

30

40

50

同様に、ソースコード 301 の全部を、そのままコンパイラに通せば、コンパイラは、`mov`、`sub`、`add`等についても、そういう識別子を持つC言語の関数として解釈するから、ソースコード 301 の全部を、何ら問題なく、コンパイル処理できる。

【0075】

しかしながら、このような取り扱いは、後述する実施の形態 2 における、コンパイラ型のシミュレータに係るものであり、実施の形態 1 における、ソフトウェア開発環境では、ライブラリ記述部を、コンパイラに通さないように、後に詳述する前処理を行う。

【0076】

また、`" /* pragma (c:r2) */ "` や `" /* pragma (d:r3) */ "` は、後述するコンパイラにおいて、変数 `c` を、ターゲットプロセッサのレジスタ `r2` に割り付け、変数 `d` を、レジスタ `r3` に割り付けるように、指定するキーワードである。

10

【0077】

図 4 は、本発明の実施の形態 1 におけるソフトウェア開発環境の機能ブロック図である。

【0078】

図 4 において、前処理部 401 は、コンパイラ 404 等の処理に先立ち、ソースコード 301 を、`" /* lib_start */ "` という開始識別子と、`" /* lib_end */ "` という終了識別子とに挟まれた、ライブラリ記述部と、それ以外の汎用記述部とに、分離する。

【0079】

そして、前処理部 401 は、ソースコード 301 の文頭から順に、汎用記述部については、コンパイラ 404 に渡して、アセンブラコードを生成させ、ライブラリ記述部については、専用トランスレータ 403 へ渡し、コンパイラ 404 を経由せずに、アセンブラコードを生成させる。これにより、アセンブラコードリスト 402 が、生成される。

20

【0080】

また、専用トランスレータ 403 は、`" mov (r2, &r3) : "` 等のソースコードを、ターゲットプロセッサ用のアセンブラコードに、一対一に変換 (`" mov (r2, r3) 関数" 1 " mov r2, r3 命令"`、`" sub (r3, r0) 関数" 1 " sub r3, r0 命令"` 等) する。

【0081】

ここで、`" /* pragma (c:r2) */ "` や `" /* pragma (d:r3) */ */ "` 等による、レジスタ割付の指示も、アセンブラコードに反映される。

30

【0082】

専用トランスレータ 403 の機能は、テキストベースの単純な置換テーブルを用いれば、容易に実現できる。

【0083】

また、ソースコード 301 の文頭で、`" lib.h "` をインクルードする、コンパイラ疑似命令が記述されているが、ライブラリ記述部をコンパイラ 404 に渡さないようにしても、このライブラリにおける関数の定義のあてはめが、なされないだけであって、コンパイラ 404 の処理上、何ら問題ない。

40

【0084】

また、このようにすることによって、後述するシミュレータにおいて、ソースコード 301 そのものを、何ら変更を加えることなく、処理できる。

【0085】

なお、ライブラリ記述部の切り分けキーワードや変数のレジスタ割り付け指定形式は、図示した例に限定されない。例えば、`" /* lib_xxx */ "` 等により、`x` 行目から `y` 行目までが、ライブラリ記述部である旨、指示することもできる。また、コンパイラが、対応できるのであれば、ライブラリ記述部の切り分けキーワードや変数のレジスタ割り付け指定形式を、コメントの形式でない形式で記述しても良い。

【0086】

50

またコンパイラ404による、C言語からアセンブラコードへの変換方法や、変数のレジスタ割り付け以外の詳細については、本発明の主旨とは関係無いことから説明を省略する。なお、本形態は、ソースコードに汎用記述部が含まれず、ソースコードの全てが、ライブラリ記述部から構成される場合にも、対応できる。

【0087】

以上のように、本形態のソフトウェア開発環境によれば、ライブラリを用いて開発したソフトウェアを翻訳しアセンブラコードを生成する際に、ライブラリ記述部は、専用トランスレータ403で、対応するアセンブラ命令に一对一変換し、汎用記述部は、コンパイラ404により、C言語で記述された処理を実現するアセンブラコードに変換することにより、ライブラリを用いたアセンブラレベルでのコード最適化を、生成するアセンブラコードに、そのまま反映できる。

10

【0088】

勿論、図4に示した、ソフトウェア開発環境に、アセンブラコードをオブジェクトコードに変換する機能を追加しても、差し支えない。

【0089】

(実施の形態2)

本形態は、ターゲットプロセッサ用のソースコード（実施の形態1のソースコード301と同じ）を読み込んで、ホストプロセッサ上で動作するオブジェクトコードを生成する、シミュレータに係る。なお、実施の形態1における説明と、重複する部分については、繰り返しを避けるため、説明を省略する。

20

【0090】

図5は、本発明の実施の形態2におけるシミュレータの機能ブロック図である。図5において、コンパイラ906は、ホストプロセッサ向けのC言語コンパイラであり、オブジェクトコード変換部503は、コンパイラ906が生成したアセンブラコード502を、オブジェクトコード504へ変換する。

【0091】

これらコンパイラ906、オブジェクトコード変換部503は、特殊なものである必要はなく、単に、ホストプロセッサ向けのものであればよい。

【0092】

但し、ソースコード301と、ライブラリ201（"lib.h"）は、実施の形態1の説明で述べた内容になっていなければならない。

30

【0093】

なおこのとき、ソースコード301において、"/* lib_start */"や"/* Pragma(c:2) */"等の表記を省略することもできるが、シミュレータと、ソフトウェア開発環境とにおいて、同一のソースコードを用いることが望ましいので、省略しない方がよい。

【0094】

また、ここでは、ソースコード301の文頭の、"lib.h"をインクルードするコンパイラ疑似命令は、省略できない。なぜなら、これを外すと、コンパイラが、ライブラリ201を参照できなくなるからである。

40

【0095】

このように、本形態によれば、コンパイラ型のシミュレータを実現でき、ホストプロセッサだけで、ターゲットプロセッサ用のソフトウェアの、シミュレーションができる。

【0096】

しかも、このシミュレータが生成し、ホストプロセッサ上で動作するソフトウェアは、インタープリタ型のそれよりも、遙かに高速であり、実際のターゲットプロセッサ用のソフトウェアの実行速度に近い、実行速度が得られるため、より効果的な検証を行える。

【0097】

また、ターゲットプロセッサのアセンブラレベルでの命令コード機能を、C言語等の高級言語で実現するライブラリを用意することで、アセンブラレベルでのコード最適化（コー

50

ドサイズ削減や実行時間の高速化等)を含めたソフトウェアを、全てC言語等の高級言語で開発できる。

【0098】

これは、従来アセンブラ言語を用いて開発されたソフトウェアを他のプロセッサへ移植する場合、アセンブラ言語で記述された部分を移植先のプロセッサ向けに修正する必要があったのに対し、Cコンパイラに対応したプロセッサであれば、そのまま何の修正も無しに移植が可能であり、ソフトウェアの汎用性を高めるものである。

【0099】

(実施の形態3)

以下、実施の形態3について、図5～図10を参照しながら説明する。実施の形態3では、割り込み及び実行サイクル数への対応を、追加する。 10

【0100】

図6は、本発明の実施の形態3におけるシミュレータの機能ブロック図である。

【0101】

このシミュレータを概説する。ライブラリ501は、ターゲットプロセッサのアセンブラレベルの命令コードの機能をC言語で定義した、関数群を含む。

【0102】

また、ターゲットプロセッサ向けソースコード301は、このライブラリ501において定義された関数等を用いて記述したライブラリ記述部と、それ以外の汎用記述部とを含む。 20

【0103】

また、アセンブラコード変換部507は、C言語のコンパイラを備え、このコンパイラで、ライブラリ記述部及び汎用記述部を、コンパイルし、ホストプロセッサ上で動作するアセンブラコード508を生成する。

【0104】

さらに、オブジェクトコード変換部509は、生成されたアセンブラコード508を、オブジェクトコードに変換し、ホストプロセッサ上で実行可能なオブジェクトコード(α.ouて*)504が生成され、このコード504が、ホストプロセッサ上で実行される。

【0105】

以下、図7を用いて、本形態で取り上げるターゲットプロセッサの構成を、具体的に説明する。図7は、本発明の実施の形態3で取り上げるターゲットプロセッサのブロック図である。 30

【0106】

このターゲットプロセッサは、基本的に、実施の形態1に係る図1(α)に示した構成と同様の構成を持つが、16ビットレジスタ17が追加されている点異なる。

【0107】

このレジスタ17の値は、通常「0」であるが、加減算器601で演算結果が符号付16ビットデータの最大値(0×7FFF)よりも大きい値になった場合又は符号付16ビットデータの最小値(0×8000)よりも小さい値になった場合に「1」にセットされる。 40

【0108】

そして、レジスタ17が「1」にセットされると、実行中のプログラムに割り込みが発生して、レジスタ17は「0」にリセットされる。

【0109】

割り込み発生後のシーケンスとしては、割り込み処理関数(本形態では、関数名をintS())とする)に制御を移す。

【0110】

そして、割り込み処理関数の処理が終了すると、元のプログラムに復帰し処理を続行する。

【0111】

ここで、割り込み発生時のターゲットプロセッサのハードウェア的なシーケンスの詳細については、本発明の主旨とは関係無いことから説明を省略する。

【0112】

図8は、図6のライブラリ全文をC言語で記述した例示図である。

【0113】

このライブラリは、基本的に、実施の形態1のもの（図2）と同様であるが、ターゲットプロセッサが備える割り込み機能並びにソフトウェアの実行サイクル数カウント機能が追加されている点が異なる。また、このライブラリファイルの名前は、“lib.h”である。

【0114】

10

このライブラリは、割り込み機能に関し、16ビットレジスタit機能を実現する変数itの定義（“static it=0;”の部分）と、各アセンブラ命令（mov命令、add命令、sub命令）の機能を実現する関数群（mov関数、add関数、sub関数）とを備える。

【0115】

また、これらの関数は、その先頭に、変数itの値が0ではなかったら変数itの値を0にリセットし、割り込み処理関数（its（））を呼ぶ機能（“if（it）{”の部分）を備える。

【0116】

また、add関数及びsub関数は、演算結果が符号付16ビットの最大値（0×7FFF）より大きい場合又は符号付16ビットの最小値（0×8000）より小さい場合に、変数itに1をセットする機能（“it=1;”の部分）を備える。

20

【0117】

ここで、ライブラリ内における上記割り込み機能に関する処理を実行するかどうかは、当該処理を条件付コンパイラ疑似命令“#ifdef _IR”及び“#endif”で指定しており、後述するコンパイル時に選択可能である。

【0118】

また、このライブラリは、ソフトウェアの実行サイクル数カウント機能に関し、サイクル数カウント用の変数countの定義（初期値は0）（“int count=0;”の部分）を備える。また、各関数（mov関数、add関数、sub関数）は、変数countをインクリメントする機能（“count+=1;”の部分、ここで変数countをインクリメントしている値（1）は各関数が機能を実現するアセンブラ命令（mov命令、add命令、sub命令）をターゲットプロセッサ上で実行した場合の処理サイクル数である）を備える。

30

【0119】

図9は、図6のアセンブラコード変換部の内部を示す詳細図である。図9に示すように、アセンブラコード変換部507は、まずデータ接続用コード生成部905を用いて、接続コード挿入済みソースコード908を生成する。

【0120】

本形態における、接続コード挿入済みソースコード908は、ソースコード301に対し、ソースコード301のライブラリ記述部と汎用記述部の間に、接続コード（r2=c; r3=d;等）を、挿入したものである。

40

【0121】

ここで、ソースコード301において、ライブラリ記述部では、ライブラリ501で定義されている変数（r0～r3）が使用され、汎用記述部では、通常の変数（a～d）が使用されている。

【0122】

実施の形態1において、このコード908を、ターゲットプロセッサ向けに翻訳する際、変数（r0～r3）は、専用トランスレータにより、対応するレジスタ（r0～r3）に割り付けられ、また変数（a～d）もコンパイラによりレジスタ（r0～r3）のいずれ

50

かに自動的に割り付けられる。

【0123】

このとき、実施の形態1において、ライブラリ記述部で使用しているレジスタと汎用記述部で使用している変数($\alpha \sim d$)の対応付け(どのレジスタにどの変数の値が格納されるか)を意識し、変数($\alpha \sim d$)を、どのレジスタに割り付けるかを指定しておけば、アセンブラコードにおけるレジスタ間でのデータの整合性がとられ、意図した処理が実行できる。

【0124】

しかしながら、実施の形態2では、ソースコード301ではなく、接続コード挿入済みソースコード908を、コンパイラを用いて翻訳する。

10

【0125】

以下、データ接続用コード生成部905により、ソースコード301に接続コードを挿入する点について、説明する。ここで、実施の形態3におけるシミュレータで、ソースコード301を、そのままコンパイラ906でコンパイルすると、ライブラリ記述部で使用している変数($r0 \sim r3$)と、汎用記述部で使用している変数($\alpha \sim d$)とが、全く別の変数として扱われてしまい、データの整合性がとれないことになる。

【0126】

そこで、実施の形態3では、データ接続用コード生成部905において、ライブラリ記述部における変数($r0 \sim r3$)と、汎用記述部における変数($\alpha \sim d$)との対応付け情報を基に、ソースコード301にライブラリ記述部と汎用記述部間の接続コード($r2 = c$; $r3 = d$; 等)を挿入し、接続コード挿入済みソースコード908を生成し、このソースコード908を、コンパイラ906でコンパイルすることとしている。

20

【0127】

以下、データ接続用コード生成部905の動作を具体的に説明する。図10は、本発明の実施の形態3におけるデータ接続用コード生成部のフローチャートである。

【0128】

まず、データ接続用コード生成部905は、ステップ1001において、ライブラリを用いたソースコード301からライブラリ記述部における変数($r0 \sim r3$)と汎用記述部における変数($\alpha \sim d$)の対応付け情報を取得する。

【0129】

図6に示すソースコード301では、この変数対応付け情報は、" Pragma (c:r2) * / "、" Pragma (d:r3) * / "として与えられている。

30

【0130】

ステップ1001では、ソースコード内を" Pragma "というキーワードより検索し、このキーワードの後の" $()$ "内の" $:$ "で区切られたパラメータを、変数対応付け情報とする。

【0131】

ここでは、変数 c と変数 $r2$ 、変数 d と変数 $r3$ が対応付けられているものとする。なお、変数対応付け情報の与え方は、図示した例に限定されない。

40

【0132】

次に、ステップ1002において、データ接続用コード生成部905は、ライブラリを用いたソースコード301におけるライブラリ記述部の場所を解析する。

【0133】

図6に示すソースコード301に対し、データ接続用コード生成部905は、" lib_start * / "という開始識別子の位置を、ライブラリ記述部のスタートポイントとして認識し、" lib_end * / "という終了識別子の位置を、ライブラリ記述部のエンドポイントと認識する。

【0134】

即ち、ステップ1002では、データ接続用コード生成部905は、ソースコード301

50

内を"lib-seta"、"lib-end"というキーワードで検索し、ライブラリ記述部の場所(スタートポイントとエンドポイント)を解析する。

【0135】

次に、ステップ1003において、データ接続用コード生成部905は、ステップ1002で特定した、ライブラリ記述部のスタートポイントに、ステップ1001で取得した変数対応付け情報に基づいて、接続コードを挿入する。

【0136】

挿入される接続コードは、変数対応付け情報で指定されたライブラリ記述部における変数(r0~r3)に、汎用記述部における変数(a~d)のデータを転送するコードである。図9の例では、変数cと変数r2、変数dと変数r3が対応付けられており、"r2=c;"、"r3=d;"が挿入される。

10

【0137】

次に、ステップ1004において、データ接続用コード生成部905は、ステップ1002で特定した、ライブラリ記述部のエンドポイントに、ステップ1001で取得した変数対応付け情報に基づいて、接続コードを挿入する。

【0138】

挿入される接続コードは、変数対応付け情報で指定された汎用記述部における変数(a~d)に、ライブラリ記述部における変数(r0~r3)のデータを転送するコードである。図9の例では、"c=r2;"、"d=r3;"が挿入される。

20

【0139】

以上のようにして、データ接続用コード生成部905は、ソースコード301に対し、上記接続コードを挿入し、接続コード挿入済みソースコード908を生成する。

【0140】

次に、コンパイラ906は、ライブラリ501及びソースコード908を翻訳し、ホストプロセッサのアセンブラコード903を生成し、これを、オブジェクトコード変換部509が、実行可能なオブジェクトコード904に変換する。

【0141】

本発明が予定するコンパイラは、特殊なものである必要はなく、ホストプロセッサに対応した汎用のコンパイラで十分である。例えば、パソコンやワークステーション向けのFree Software FoundationのCコンパイラGCC等を用いることができる。

30

【0142】

ここで、Cコンパイラ(GCC)を用いてソースコードを翻訳し、アセンブラコードを生成する際に、オプションとして"-D-IR"を指定すれば、上記割り込み機能を組み込むことができ、また指定しなければ当該機能を外すことが可能である。

【0143】

最後に、図6に示すように、生成されたオブジェクトコード(a.out*)504(図5)を、ホストプロセッサ(パソコンやワークステーション等)上で実行することで、ライブラリ内で記述されたターゲットプロセッサの機能(演算、割り込み等)や実行サイクル数カウント機能もホストプロセッサ上で実現でき、ターゲットプロセッサ向けに開発したソフトウェア(アセンブラレベルも含む)のシミュレーションが可能となる。

40

【0144】

また、このシミュレータは、コンパイラ型であり、実行前にコンパイルするため、実行時に実行命令の解釈処理などが必要なく、従来のインタプリタ型シミュレータ上でのシミュレータに対して、高速なシミュレーションが可能となる。

【0145】

以上のように、本形態によれば、アセンブラレベルでのコード最適化(コードサイズ削減や実行時間の高速化等)を含めたソフトウェアを全てC言語等の高級言語で開発し、開発したソフトウェアをホストプロセッサ上でコンパイル、実行でき、従来のシミュレータによるシミュレーションよりも、高速なシミュレーションが可能となる。

50

【0146】

また、ライブラリ記述部で使用している変数($r_0 \sim r_3$)と、それ以外の部分で使用している変数($a \sim d$)の対応付けに対応することにより、ライブラリ記述部と汎用記述部において、データの整合性を担保できる。

【0147】

また、ライブラリ内にターゲットプロセッサの割り込み機能を備え、かつシミュレーション実行時に、この機能を使用するかどうかが選択可能とすることにより、割り込み機能が必要な場合、当該機能を使用したシミュレーションを行い、逆に、割り込み機能が不要な場合、当該機能を使用せず高速なシミュレーションを行える。

【0148】

また、ライブラリ内で用意したサイクル数カウンタ変数 `count` を、例えばシミュレーション終了時にC言語における標準出力関数 `printf` 等で表示させたりすることにより、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照可能となる。

【0149】

【発明の効果】

本発明のソフトウェア開発環境によれば、アセンブラレベルでのコード最適化を、生成するアセンブラコードにそのまま反映できる。

【0150】

また、本発明のシミュレータによれば、従来のシミュレータによるシミュレーションよりも、高速なシミュレーションを行える。

【0151】

また、ライブラリ記述部と汎用記述部の、変数の対応付けに対応して、これらの間における、データの整合性を担保できる。

【0152】

また、ターゲットプロセッサの割り込み機能を備え、割り込み機能のオン/オフに対応して、割り込み機能が必要な場合は当該機能を使用したシミュレーションを行い、割り込み機能が不要なシミュレーションの場合は高速なシミュレーションを行える。

【0153】

また、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照できる。

【図面の簡単な説明】

【図1】(a) 本発明の実施の形態1で取り上げるターゲットプロセッサのブロック図
(b) 同ターゲットプロセッサ上で実行可能なアセンブラレベルの命令及びその機能を示す説明図

【図2】図1のターゲットプロセッサに対応するライブラリをC言語で記述した例示図

【図3】本発明の実施の形態1におけるソースコードの例示図

【図4】本発明の実施の形態1におけるソフトウェア開発環境の機能ブロック図

【図5】本発明の実施の形態2におけるシミュレータの機能ブロック図

【図6】本発明の実施の形態3におけるシミュレータの機能ブロック図

【図7】本発明の実施の形態3で取り上げるターゲットプロセッサのブロック図

【図8】図6のライブラリ全文をC言語で記述した例示図

【図9】図6のアセンブラコード変換部の内部を示す詳細図

【図10】同データ接続用コード生成部のフローチャート

【図11】(a) 従来のソースコードの例示図

(b) 従来のソースコード及びアセンブラコードの例示図

【符号の説明】

201、501 ライブラリ

301 ソースコード

401 前処理部

10

20

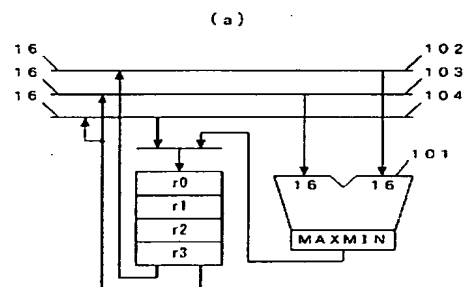
30

40

50

403 専用トランスレータ
 404、906 コンパイラ
 503、509 オブジェクトコード変換部
 507 アセンブラコード変換部

【図1】



(b)

命令	mov	add	sub
記法形式	mov src, dst	add src, dst	sub src, dst
機能	dst ← src	dst ← src + dst	dst ← src - dst
備考	srcは[r0, r1, r2, r3]中から選択 dstは[r0, r1, r2, r3]中から選択		

【図2】

lib.h 201

```

/* レジスタ機能を実現する変数の定義 */
short r0, r1, r2, r3;

/* アセンブラ命令の機能を実現する関数の定義 */
/* mov命令 */
void mov(short src, short *dst) {
    /* 転送処理 */
    *dst = src;
}

/* add命令 */
void add(short src, short *dst) {
    int tmp;

    /* 加算処理 */
    tmp = (int)src + (int)*dst;

    /* MAXMIN処理 */
    if (tmp > (int)0x0000FFFF) {
        *dst = (short)0x7FFF;
    }
    else if (tmp < (int)0xFFFF8000) {
        *dst = (short)0x8000;
    }
    else {
        *dst = (short)tmp;
    }
}

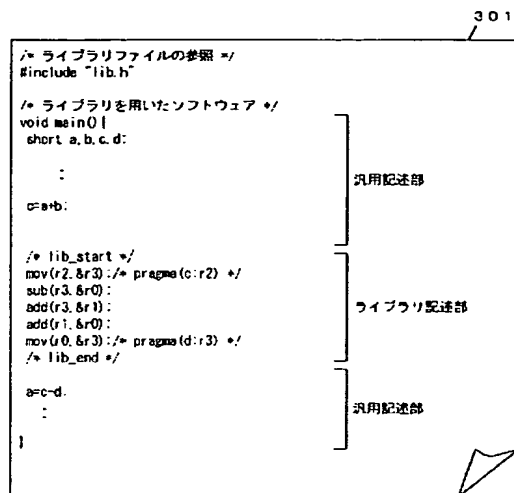
/* sub命令 */
void sub(short src, short *dst) {
    int tmp;

    /* 減算処理 */
    tmp = (int)src - (int)*dst;

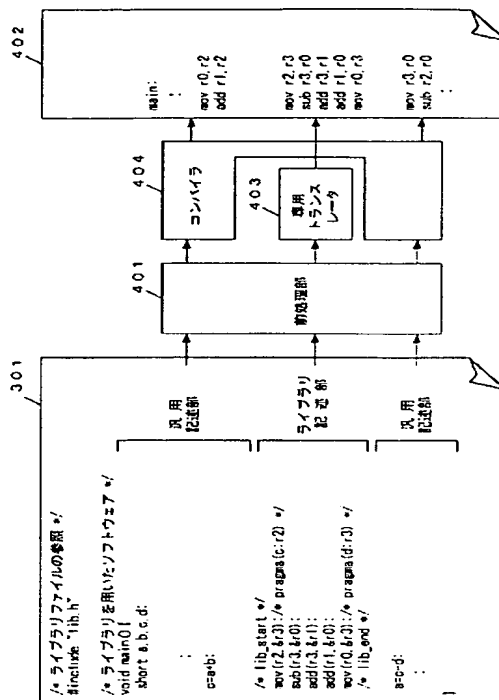
    /* MAXMIN処理 */
    if (tmp > (int)0x0000FFFF) {
        *dst = (short)0x7FFF;
    }
    else if (tmp < (int)0xFFFF8000) {
        *dst = (short)0x8000;
    }
    else {
        *dst = (short)tmp;
    }
}

```

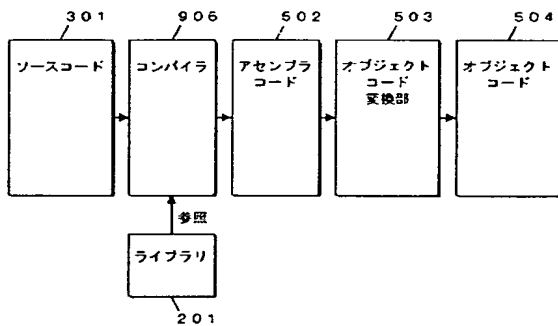
【図 3】



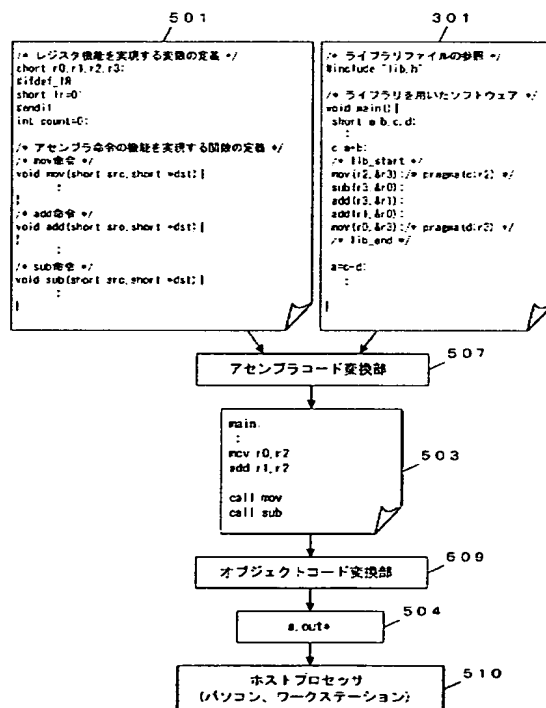
【図 4】



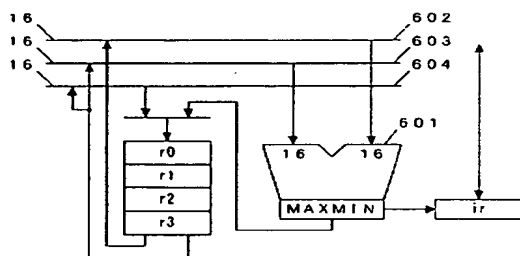
【図 5】



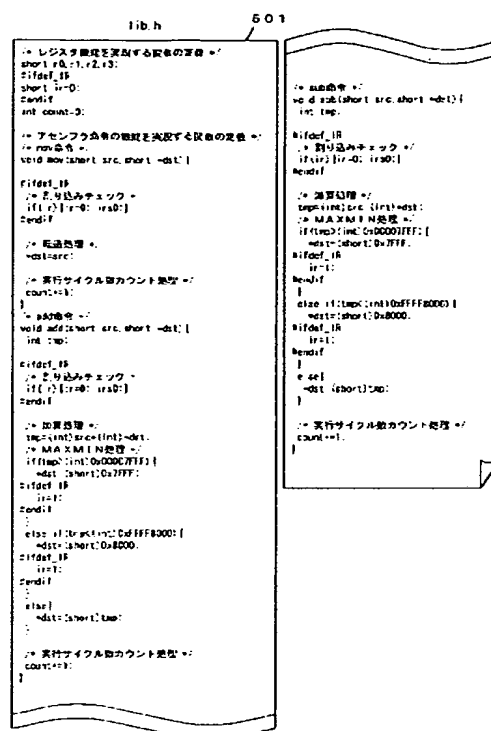
【図 6】



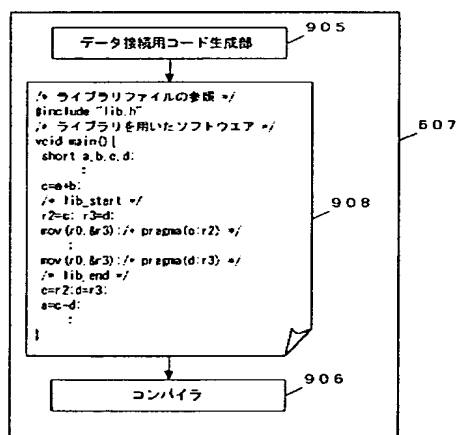
【図 7】



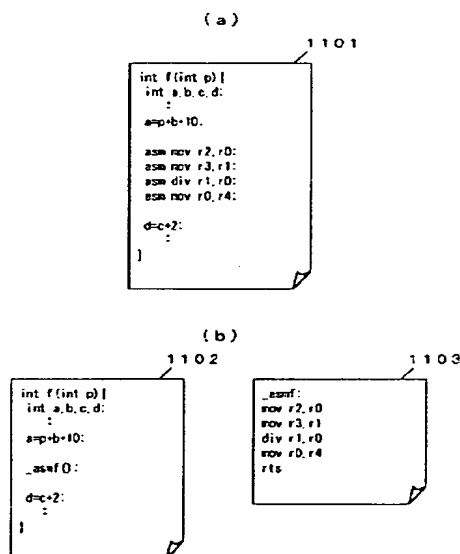
【図 8】



【図 9】



【図 11】



【図 10】

